Application programming on parallel/distributed computing platforms

Daniele Lezzi - BSC

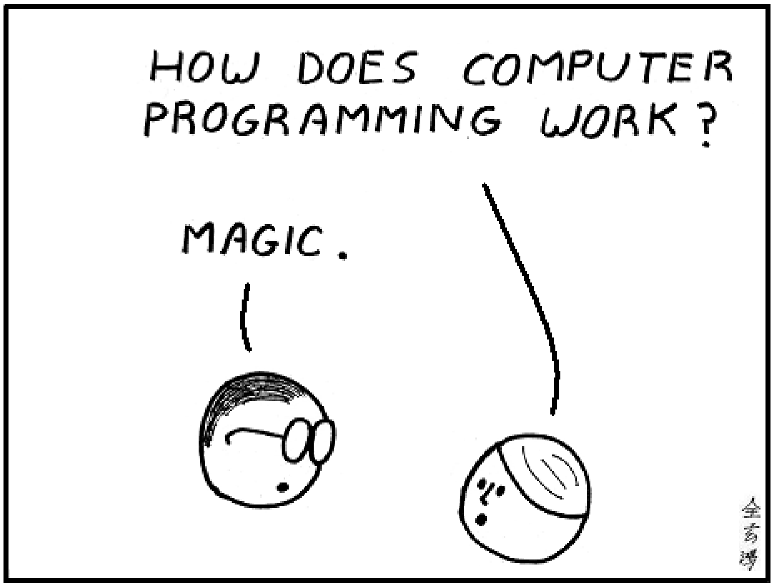
11/01/2018

Training week - Munich

Outline

- Programming parallel and distributed computing platforms: an overview
- Programming in PyCOMPSs/COMPSs
- Resource management and COMPSs Runtime

Challenges



Computation platforms

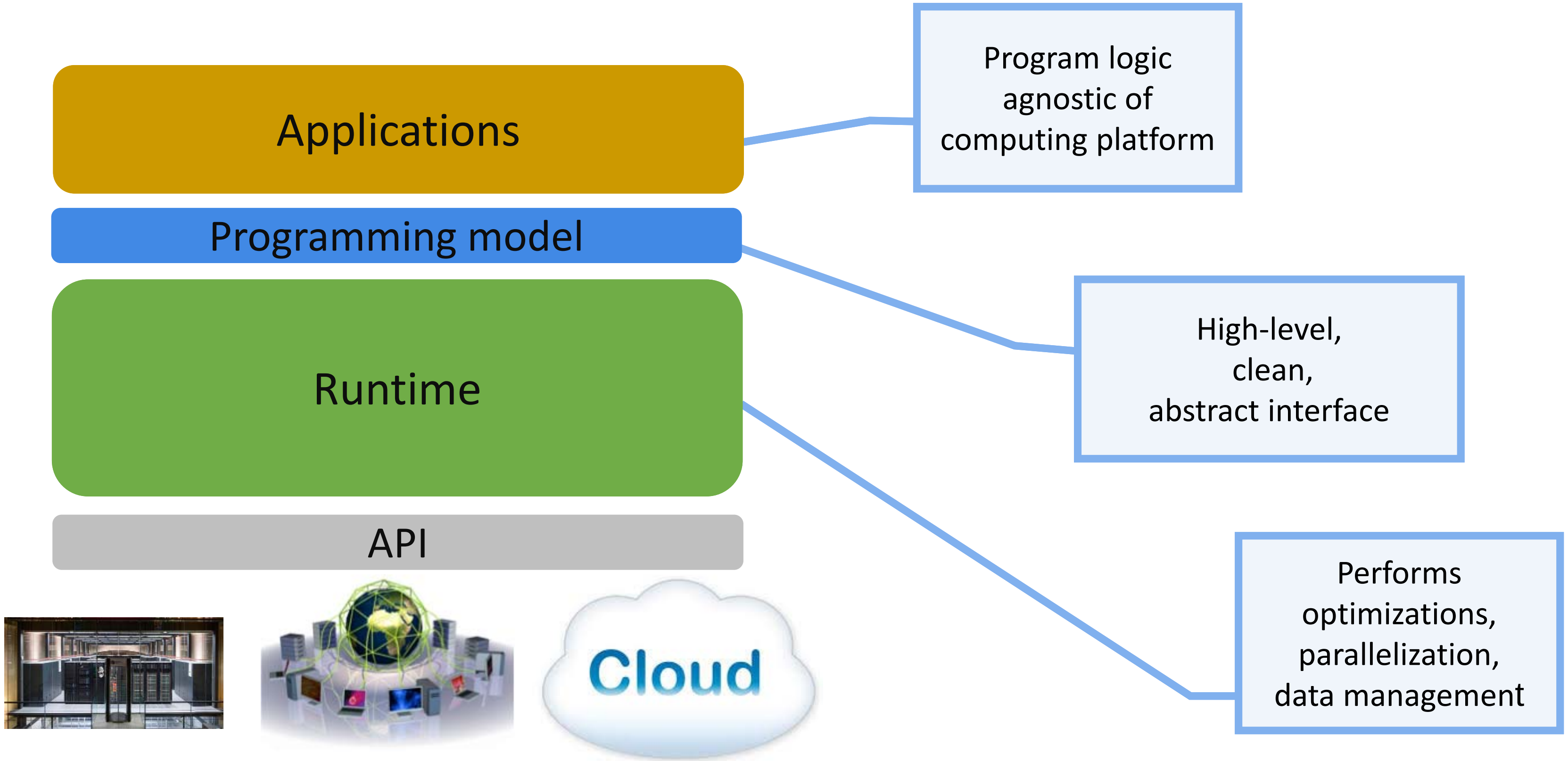
- New architectures and organization of processors
  - Multicore
    - Including vector units
  - GPU/accelerators
  - FPGAs

- Shift on programming paradigms:
  - From sequential to parallel
  - New instructions/languages
- Computing paradigms:
  - From Clusters, through Grids, to Cloud

Why is difficult to program distributed environments?

- Gap between traditional way of programming and actual hardware
  - Multicore
  - Heterogeneity
  - Distribution
    - Multiple nodes
    - Distributed memory systems
    - Manifold middleware to manage the resources (cloud, containers, ...)
- A lot of applications are thought sequential and for shared memory and then ported to distributed environments

BSC vision on programming models



Applications

Programming model

Runtime

API

Program logic agnostic of computing platform

High-level,
clean,
abstract interface

Performs optimizations,
parallelization,
data management

Review of related approaches

Overview of parallel programming models

- Traditional HPC distributed parallel programing
  - MPI
- Big-data programming
  - MapReduce
  - Spark
- Task-based programming

Message passing

- MPI is the largest used standard
  - MPI is an industry standard model for parallel programming
  - A large number of implementations of MPI exist (both commercial and public domain)
  - Virtually every system in the world supports MPI
- Based on explicit communication between processes
- Processes may have multiple threads sharing a single address space. MPI is for communication among processes, which have separate address spaces
- Inter-process communication consists of
  - Synchronization
  - Movement of data from one process's address space to another's.

Message passing

- Explicit calls to MPI interface
- Simple code for two processes that sends/receives the data buffer

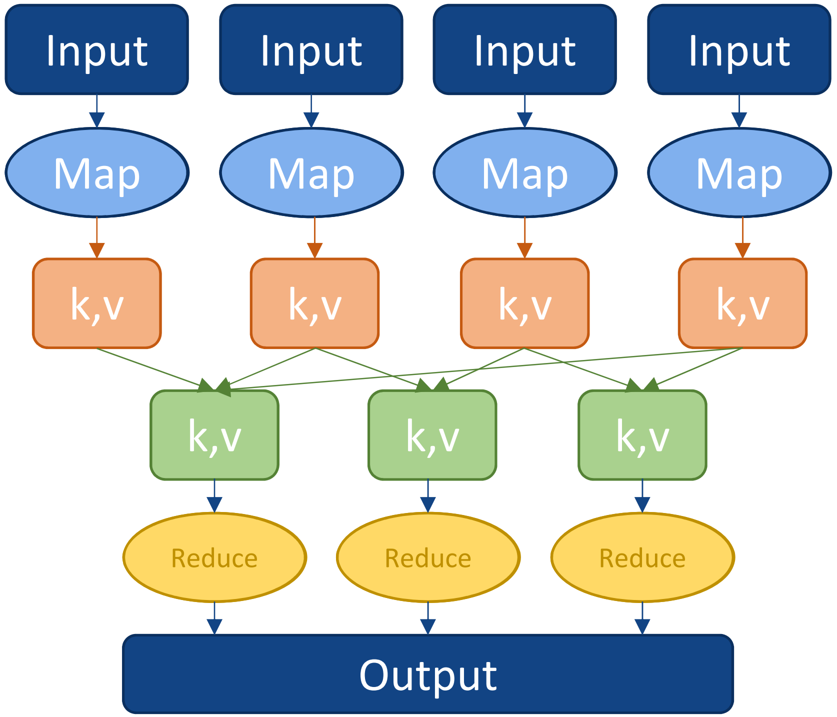
```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char ** argv)
{
    int rank, data[100];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
         MPI_Send(data, 100, MPI_INT, 1, 0, MPI_COMM_WORLD);
    else if (rank == 1)
         MPI_Recv(data, 100, MPI_INT, 0, 0, MPI_COMM_WORLD,
                  MPI_STATUS_IGNORE);
    MPI_Finalize();
    return 0;
}
```

- How to run an MPI application:

  mpirun -np 2 hello

MapReduce

- The MapReduce algorithm contains two important tasks: Map and Reduce
  - The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs)
  - The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples
- The reduce task is always performed after the map job
- Basic data structure: key-value pairs
- Storage: Hadoop Distributed File System (HDFS)



MapReduce

- WordCount example

```
public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
{
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens())
    {
         word.set(itr.nextToken());
         context.write(word, one);
    }
}
```

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException
{
    int sum = 0;
    for (IntWritable val : values)
    {
         sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
}
```

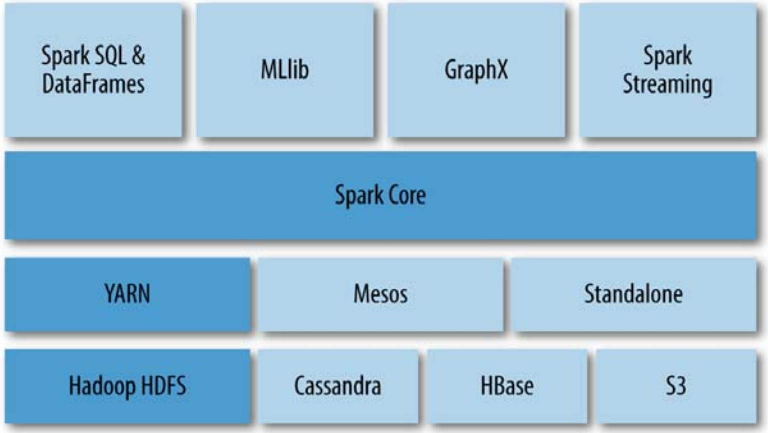
MapReduce

- WordCount example

```
...
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
...
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

Programming with Spark

- Sequential programming
- General purpose programming language + operators
- Main abstraction: Resilient Distributed Dataset (RDD)
  - Collection of read-only elements partitioned across the nodes of the cluster that can be operated on in parallel
- Operators transform RDDs
  - Transformations
  - Actions
- Simple linear address space
- Builds a DAG of operators applied to the RDDs
- Somehow agnostic of computing platform
  - Enabled by the runtime for clusters and clouds
- Uses also HDFS



Spark SQL & DataFrames

MLlib

GraphX

Spark Streaming

Spark Core

YARN

Mesos

Standalone

Hadoop HDFS

Cassandra

HBase

S3

Spark

- Sample WordCount code in Scala

```
JavaRDD<String> file = sc.textFile(inputDirPath+"/*.txt");
JavaRDD<String> words = file.flatMap(new FlatMapFunction<String, String>() {
    public Iterable<String> call(String s) {
         return Arrays.asList(s.split(" "));
    }
});
JavaPairRDD<String, Integer>
pairs = words.mapToPair(new PairFunction<String, String, Integer>() {
    public Tuple2<String, Integer> call(String s) {
         return new Tuple2<String, Integer>(s, 1);
    }
});
JavaPairRDD<String, Integer>
counts = pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {
    public Integer call(Integer a, Integer b) {
         return a + b;
    }
});
counts.saveAsTextFile(outputDirPath);
```

Task-based programming models

- The task is the basic unit for parallelism. Receives inputs, computes, generates outputs
- An application is composed of tasks
- Tasks can run in parallel
  - When data dependencies are considered, a task can only be executed once its input parameters are available
- Examples:
  - OpenMP from version 4.0
  - StarPU
  - StarSs: OmpSs and PyCOMPSs/COMPSs

www.bsc.es

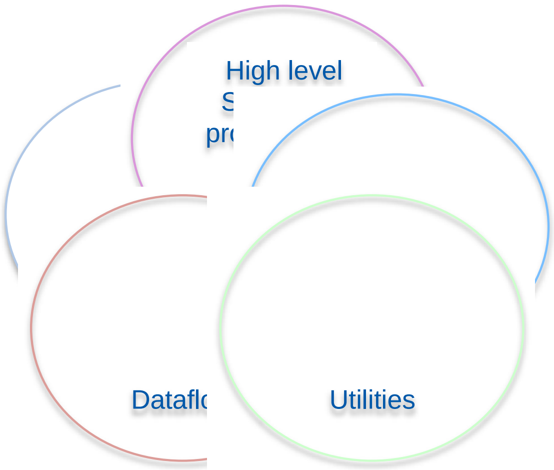
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

Introduction to PyCOMPSs/COMPSs

So, what is a superscalar programming model?

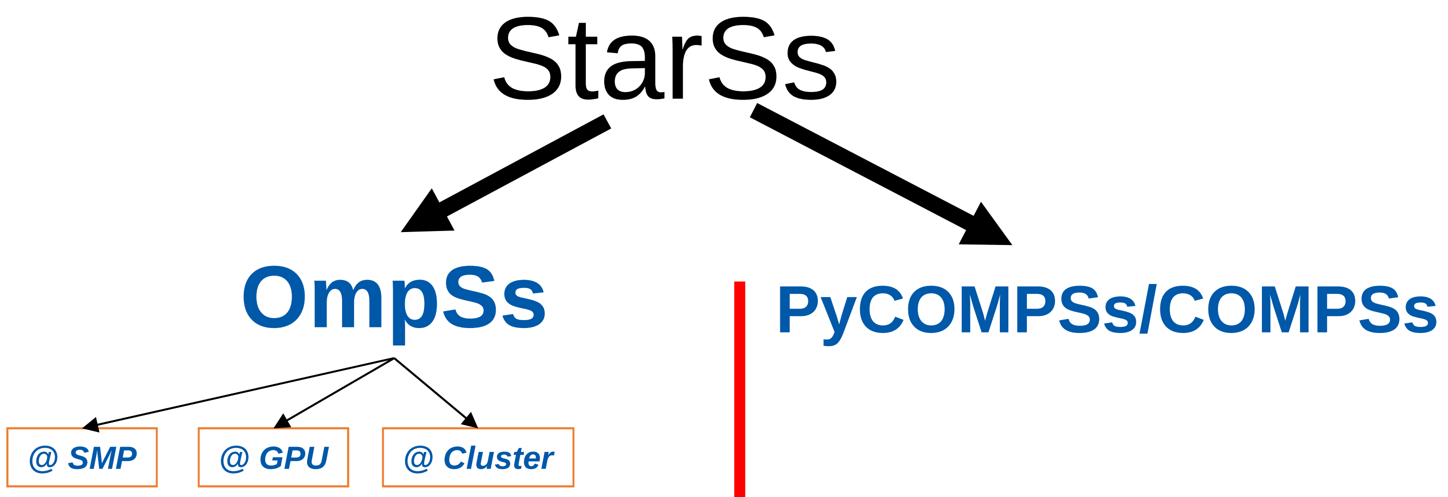
- High-level sequential programming
- Executes following superscalar processor model
  - Out of order
  - Task is the unit of work
- Builds a task graph at runtime that express potential concurrency
  - Large number of in-flight tasks
  - Exposes distant parallelism
- Based on a runtime
  - Makes decisions and executes the task-graph
  - Offers an abstraction to “plug” applications to different resources
    - Computing
    - Storage



Main elements of superscalar programming model syntax

- Superscalar program
  - Sequential code
  - Single shared memory space
  - Identification of tasks
- Task
  - Main element of programming model: computation unit
  - Operates in given parameters and local variables
  - Amount of work (granularity) may vary in a wide range (from µsecs, to minutes, hours), may depend on input arguments,…
  - Once started executes to completion independent of other tasks
- Syntax
  - Task annotations
  - Task arguments directionality
  - Synchronizations

The StarSs family

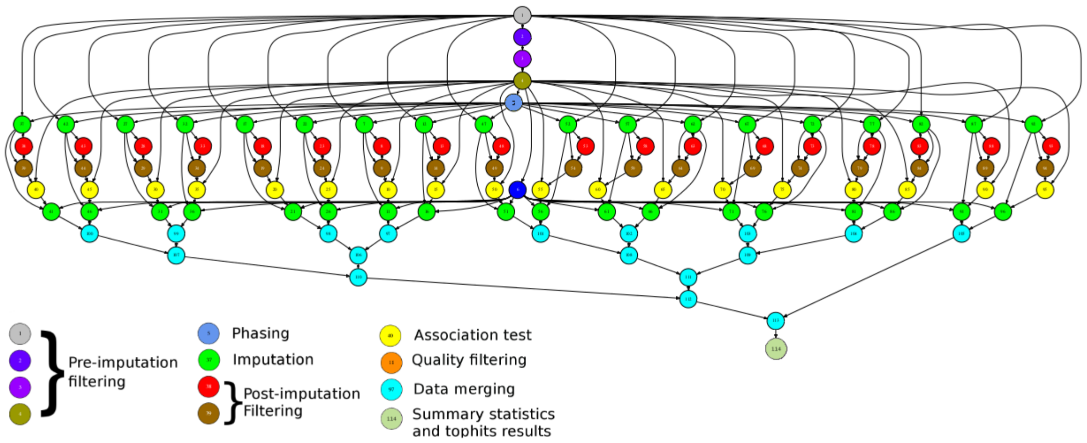


Average task Granularity:	
100 microseconds – 10 milliseconds	10 ms - 1 day
Address space to compute dependences:	
Memory	Files, Objects, NVMs
Language bindings:	
C, C++, FORTRAN	Java, C/C++, Python

SMPs, Clusters **Clusters, Clouds**

Programming with PyCOMPSs/COMPSs

- Sequential programming
- General purpose programming language + annotations/hints
  - To identify tasks and directionality of data
- Task based: task is the unit of work
- Simple linear address space
- Builds a task graph at runtime that express potential concurrency
  - Implicit workflow
- Exploitation of parallelism
  - … and of distant parallelism
- Agnostic of computing platform
  - Enabled by the runtime for clusters, clouds and grids



PyCOMPSs

- Based on regular/sequential Python code
- Use of decorators to annotate tasks and indicate arguments directionalit
- Other annotations: constraints
- Small API for data synchronization

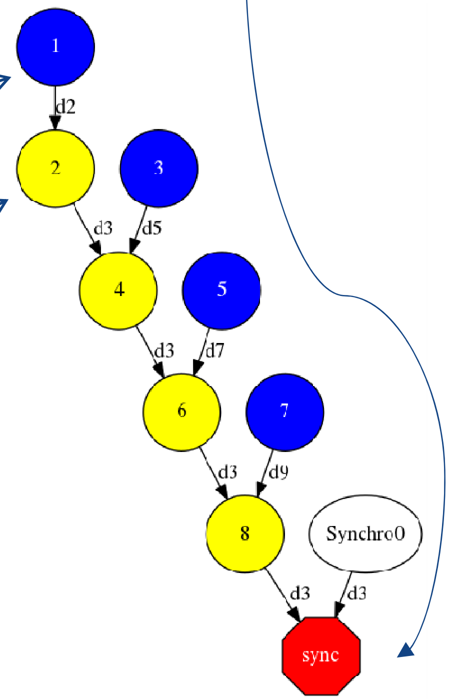
Main Program

```
Data = [block1, block2, …, blockN]
result=defaultdict(int)
for block in Data:
    presult = word_count(block)
    reduce_count(result, presult)
finalResult = compss_wait_on(result)
```

Tasks definition

```
@constraint(ProcessorCoreCount=mkl_threads)
@task(returns=dict)
def word_count(collection):
    ...
```

```
@task(dict_1=INOUT)
def reduce_count(dict_1,
dict_2):
    ...
```



PyCOMPSs Syntax

- Python decorators:

```
from pycompss.api.task import task

@task
def function():
     # do something
```

The decorator is used to indicate that the function is considered a task definition

Each call to the function will be considered as a task call

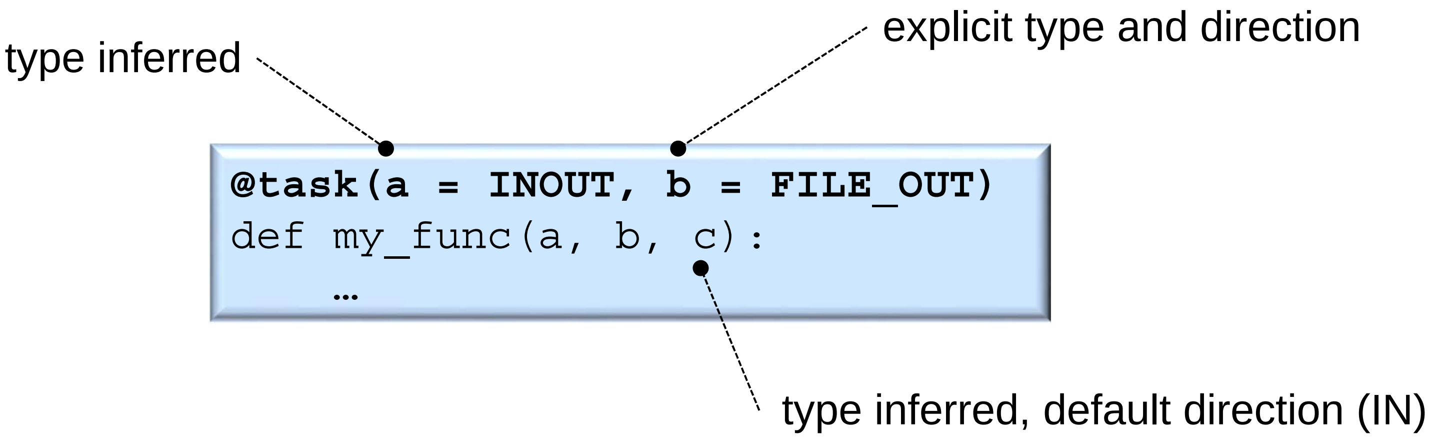
- API:

```
if __name__ == ‘__main___’:
    from pycompss.api.api import compss_wait_on

    # do some task calls

    compss_wait_on(something)
```

The API is used to indicate that a synchronization is requested

PyCOMPSs: Task definition

- Task definition with Python decorators
  - Provide information about task parameters (TYPE_DIRECTION):
    - Type
      - Only mandatory for files
      - Inferred for the rest of the types
    - Direction
      - Default IN (read-only)
      - Mandatory for INOUT (read-write) and OUT (write-only)



type inferred

explicit type and direction

```
@task(a = INOUT, b = FILE_OUT)
def my_func(a, b, c):
    …
```

type inferred, default direction (IN)

COMPSs Java

Annotated Interface

```
public interface BlastItf {
@Binary(binary = "${BLAST_BINARY}”,
          constraints = @Constraints(computingUnits = “1”))
)
Integer align(
    @Parameter(type = Type.STRING, direction = Direction.IN) String pFlag,
    @Parameter(type = Type.STRING, direction = Direction.IN) String pMode,
    @Parameter(type = Type.STRING, direction = Direction.IN) String dFlag,
    @Parameter(type = Type.STRING, direction = Direction.IN) String database,
    @Parameter(type = Type.STRING, direction = Direction.IN) String iFlag,
    @Parameter(type = Type.FILE, direction = Direction.IN) String partitionFile,
    @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
    @Parameter(type = Type.FILE, direction = Direction.OUT) String partitionOutput
);
@Method(declaringClass = "blast.BlastImpl")
void assemblyPartitions(
    @Parameter(type = Type.FILE, direction = Direction.INOUT) String partialFileA,
    @Parameter(type = Type.FILE, direction = Direction.IN) String partialFileB
);
}
```

Binary invocation

Task constraints

Regular method

Parameter metadata

Main code

```
for (int i = 0; i < numAligns; i++) {
        exitValues[i] = BINARY.align(pFlag, pMode, dFlag, Blast.databasePath,
                            iFlag, Blast.partialInputs.get(i), oFlag,
                            Blast.partialOutputs);
        BlastImpl.assemblyPartitions(Blast.Output, Blast.partialOutputs);

}
```

PyCOMPSs: Application Example (I)

- Transpose N 2D matrices
- Accumulate them

```
def transpose(matrix):
    result = [list(a) for a in zip(*matrix)]
    return result

def add(matrix1, matrix2):
    for x in range(len(matrix1)):
        for y in range(len(matrix1[0])):
            matrix1[x][y] += matrix2[x][y]
```

```
if __name__ == '__main__':
    import random
    random.seed(5)
    X = 100
    Y = 100
    min = 0
    max = 1000
    numMatrices = 500
    partialResult = [[0 for x in range(X)] for y in range(Y)]

    for i in range(numMatrices):
        matrix = [[random.randint(min, max) for x in range(X)] for y in range(Y)]
        transposed = transpose(matrix)
        add(partialResult, transposed)

    print partialResult
```

PyCOMPSs: Application Example (II)

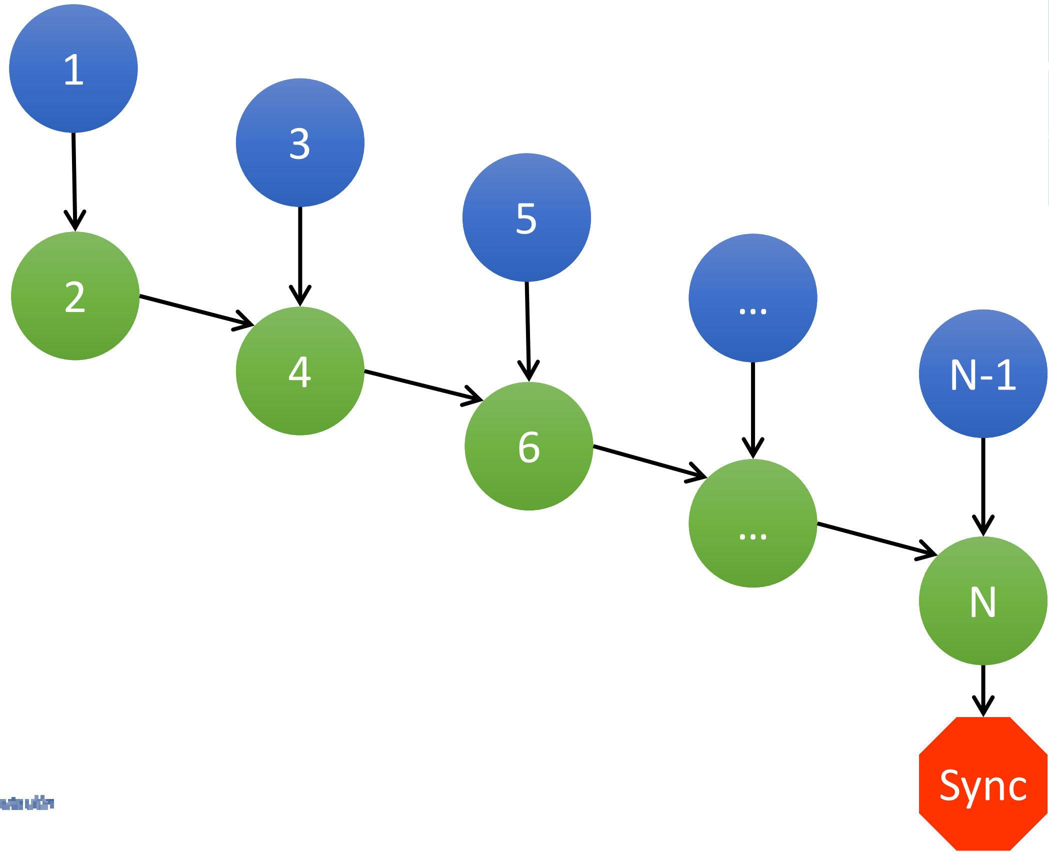
- Transpose N 2D matrices
- Accumulate them

```
from pycompss.api.task import task
from pycompss.api.parameter import *

@task(returns = list)
def transpose(matrix):
    result = [list(a) for a in zip(*matrix)]
    return result

@task(matrix1=INOUT)
def add(matrix1, matrix2):
    for x in range(len(matrix1)):
        for y in range(len(matrix1[0])):
            matrix1[x][y] += matrix2[x][y]
```

```
if __name__ == '__main__':
    from pycompss.api.api \
         import compss_wait_on
    import random
    random.seed(5)
    X = 100
    Y = 100
    min = 0
    max = 1000
    numMatrices = 500
    partialResult = [[0 for x in range(X)] for y in range(Y)]

    for i in range(numMatrices):
        matrix = [[random.randint(min, max) for x in range(X)] for y in range(Y)]
        transposed = transpose(matrix)
        add(partialResult, transposed)

    result = compss_wait_on(partialResult)
    print result
```

PyCOMPSs: Application Example (III)

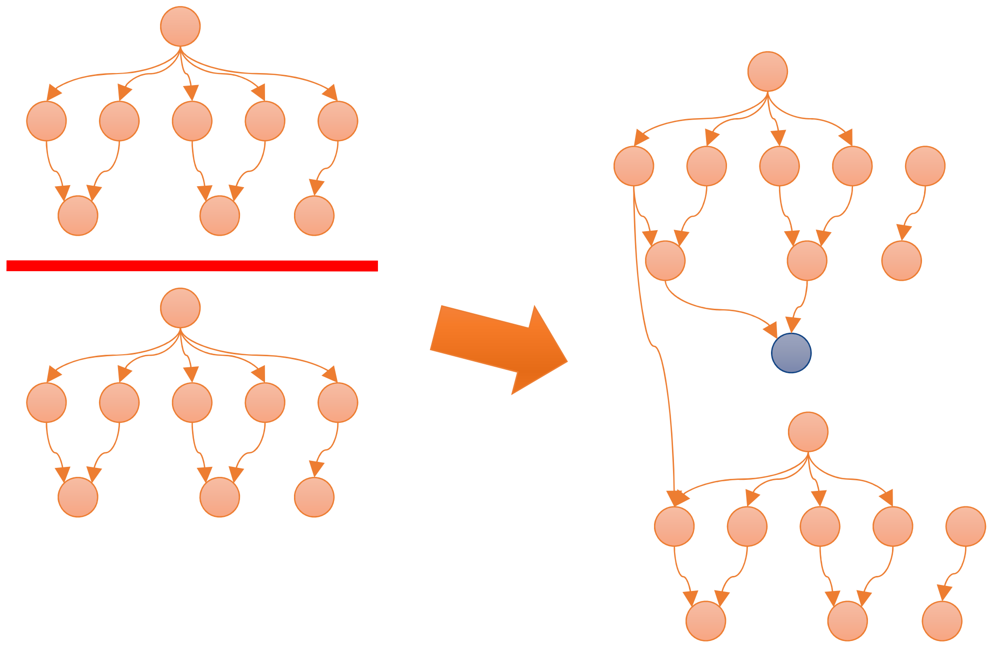
- Each matrix transpose is performed in parallel
- The result is accumulated in matrix1
- The synchronization is performed only with the accumulated variable



Color	Task
🟦	transpose
🟩	add

Programming methodology

- Tasks are the basic unit of parallelism
- Finding tasks
  - What can be a task?
    - Piece of computation with enough granularity
    - Potential for concurrency with other tasks
    - Enabler for more concurrency or tasks generation -> avoid bottlenecks



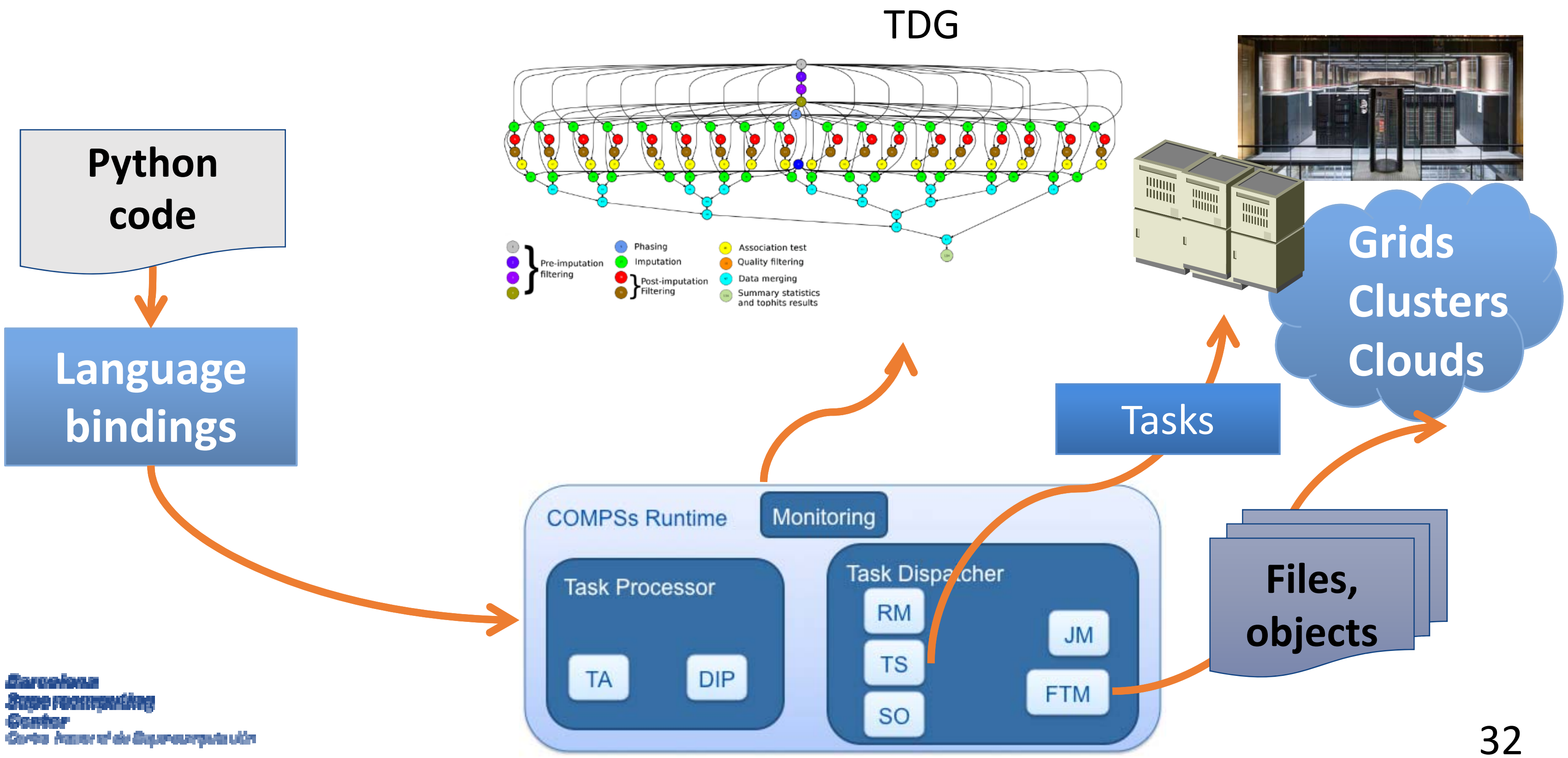
General approach for development

- Right now, no debugger available
- Methodology first Step: run serial
    - 1 single worker, 1 single task
    - Add extra synchronization points (barrier)
- Incrementally remove barriers and/or add worker tasks
- Task based monitoring
    - Visualize graph
    - Monitor execution of application in the different resources
- Several levels of logs
    - Info – generates information about file transfers and tasks execution
    - Debug – generates same information as Info level, but with much more level of detail
    - Off - no logs, only errors are reported
- Tracefile visualization and analysis - Requires that the execution finalizes
    - Can help detecting unusual behaviors: Tasks being serialized, Unexpected synchronizations...

COMPSs runtime

PyCOMPSs runtime

- Sequential execution starts in master node
- Tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers performed by runtime



TDG

Pre-imputation filtering

Phasing

Imputation

Post-imputation Filtering

Association test

Quality filtering

Data merging

Summary statistics and tophits results

Python code

Language bindings

COMPSs Runtime

Monitoring

Task Processor

TA

DIP

Task Dispatcher

RM

TS

SO

JM

FTM

Tasks

Grids

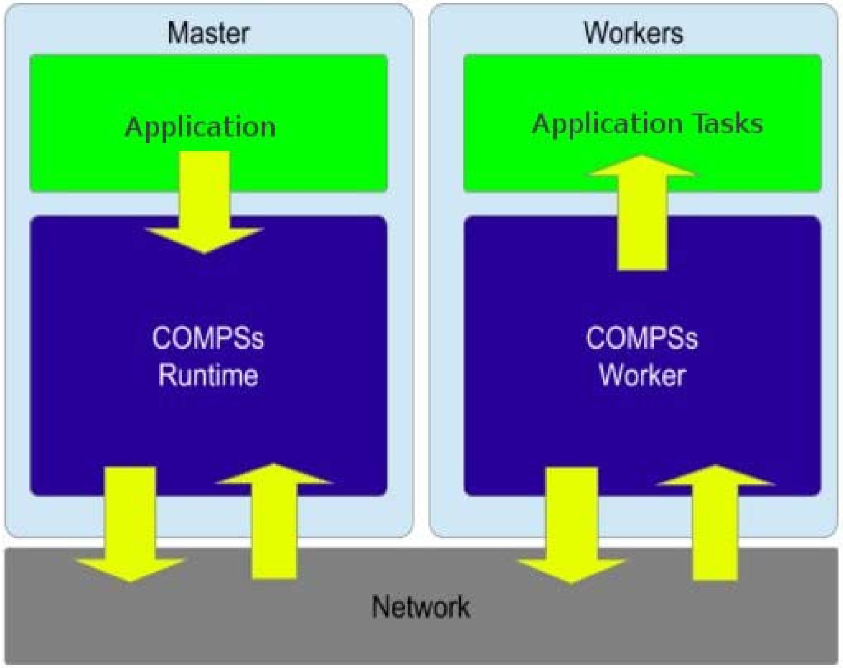
Clusters

Clouds

Files, objects

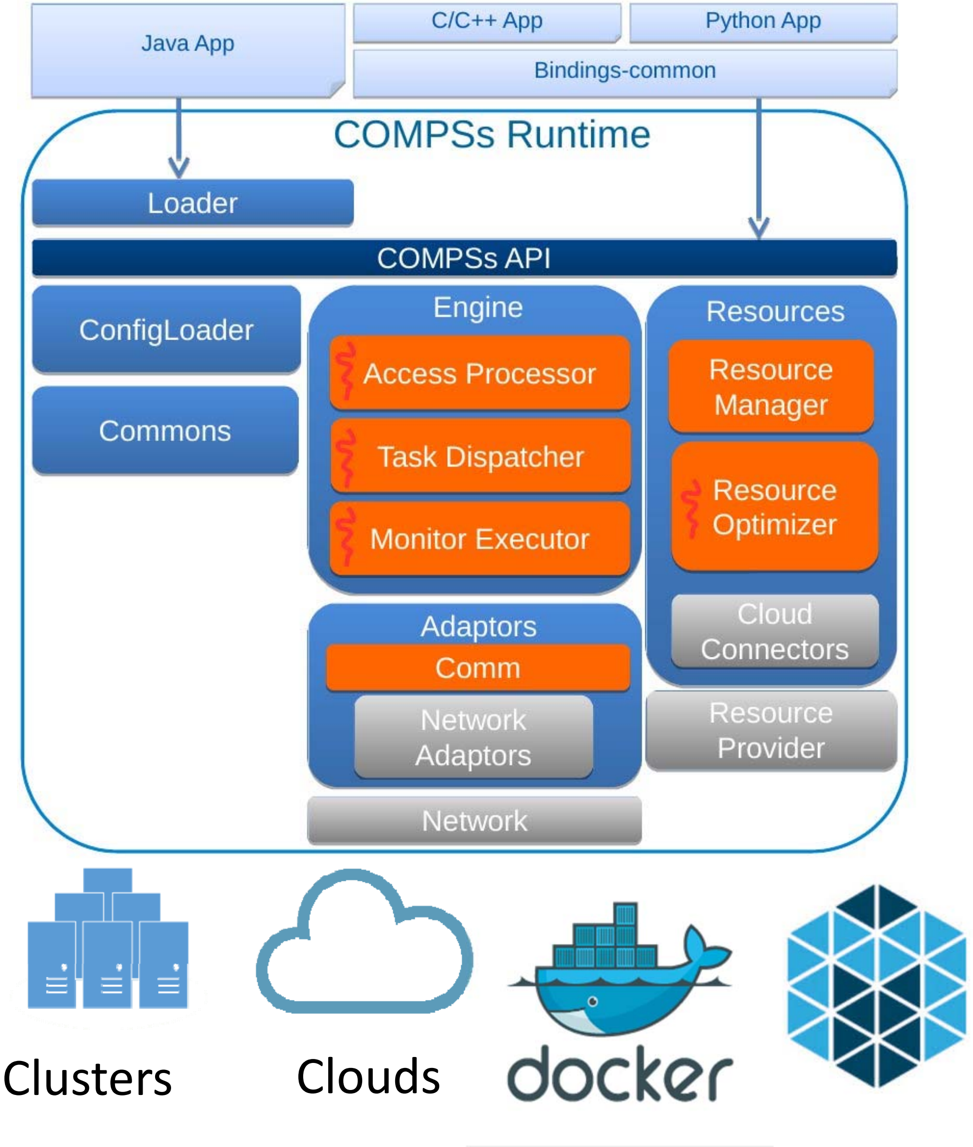
COMPSs Overview - Runtime System

- Application architecture



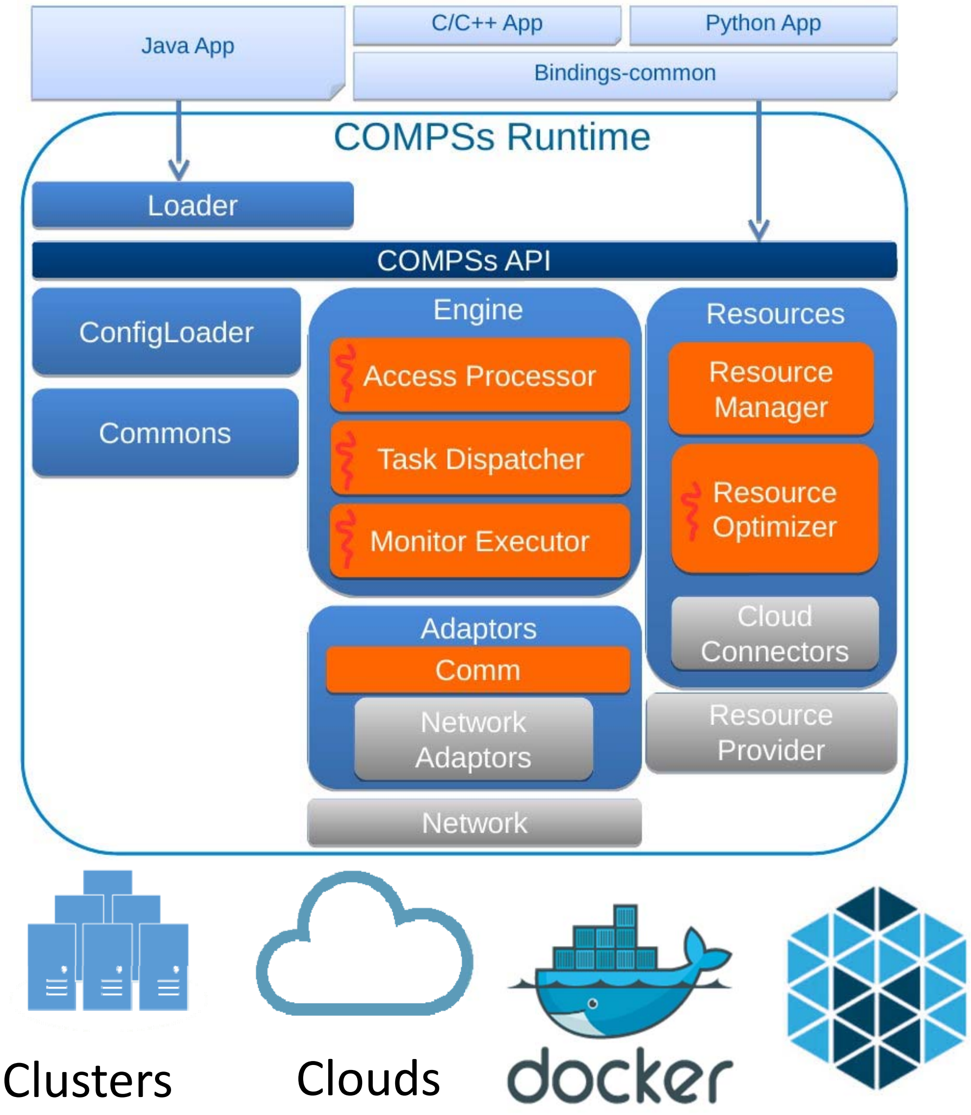
COMPSs Overview - Runtime System

- Componentized
- Adaptable
- Extensible
- Interoperable
- Each component responsible of a specific task
- Task generation
  - Dependence analysis
  - Generation of task graph
- Task scheduling
  - When?
  - To which resource?
- Data management
  - Where is the data?
  - Transfer of data

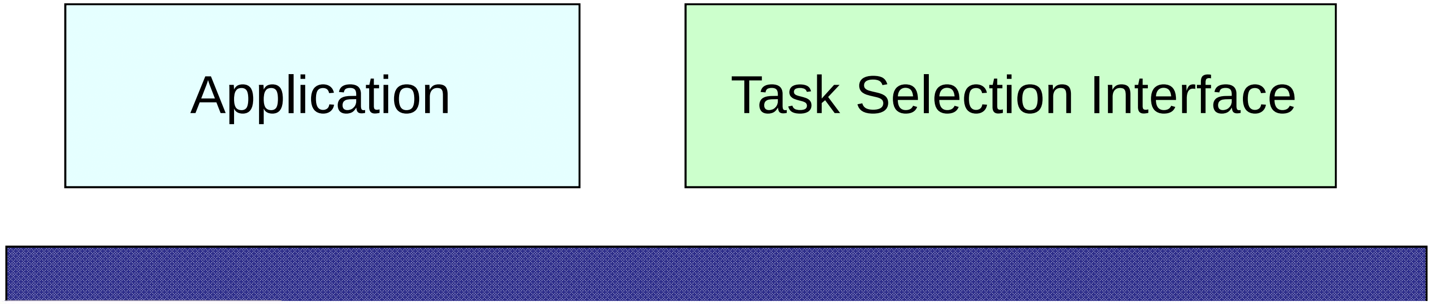


COMPSs Overview - Runtime System

- Access processor
  - Performs data management
  - Knows where data is
- Task dispatcher
  - Schedules tasks
  - Finds dependencies
  - Adds tasks to task-graph
  - Updates task-graph
- Monitor executor
  - Monitors execution at real-time
- Resource optimizer
  - Decides on creation of new machines
  - Cloud only



Runtime System



Application

Task Selection Interface

How do I select the execution platform?

Grid

Cluster

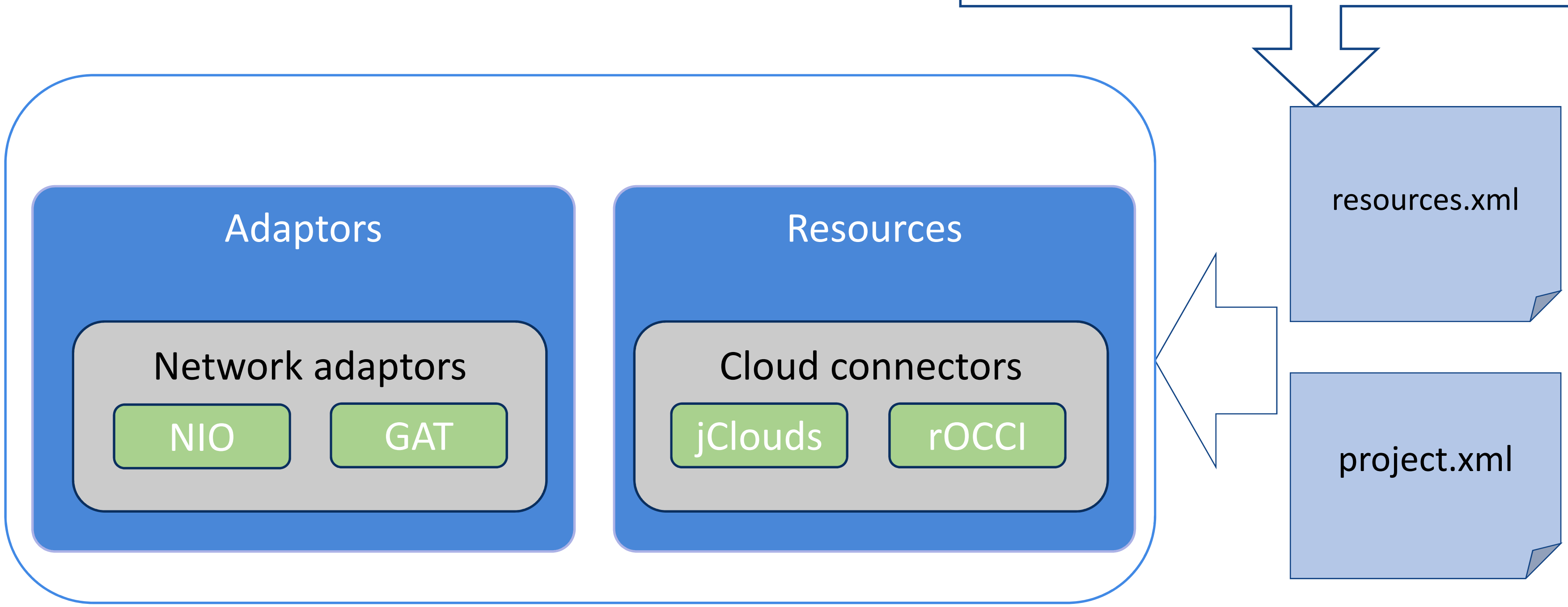
Cloud

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

COMPSs Execution Environment

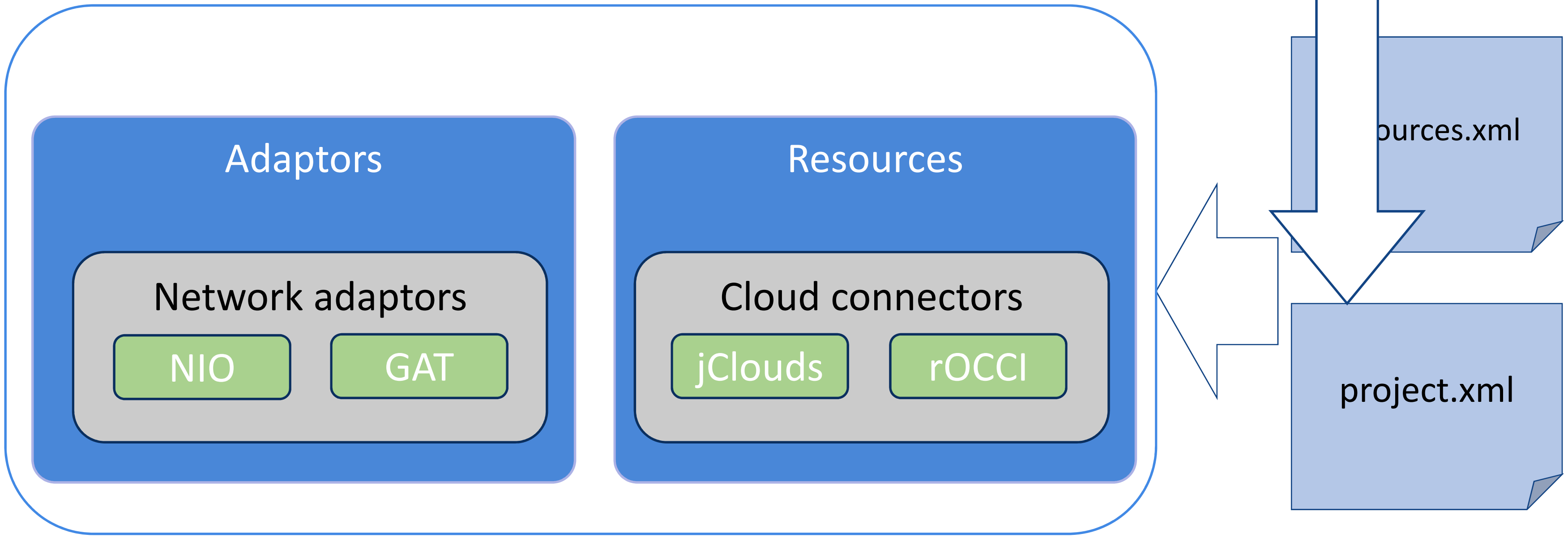
Infrastructure Description

- Describes the available resources in the infrastructure
- Describes Cloud Providers: Images and VM Templates

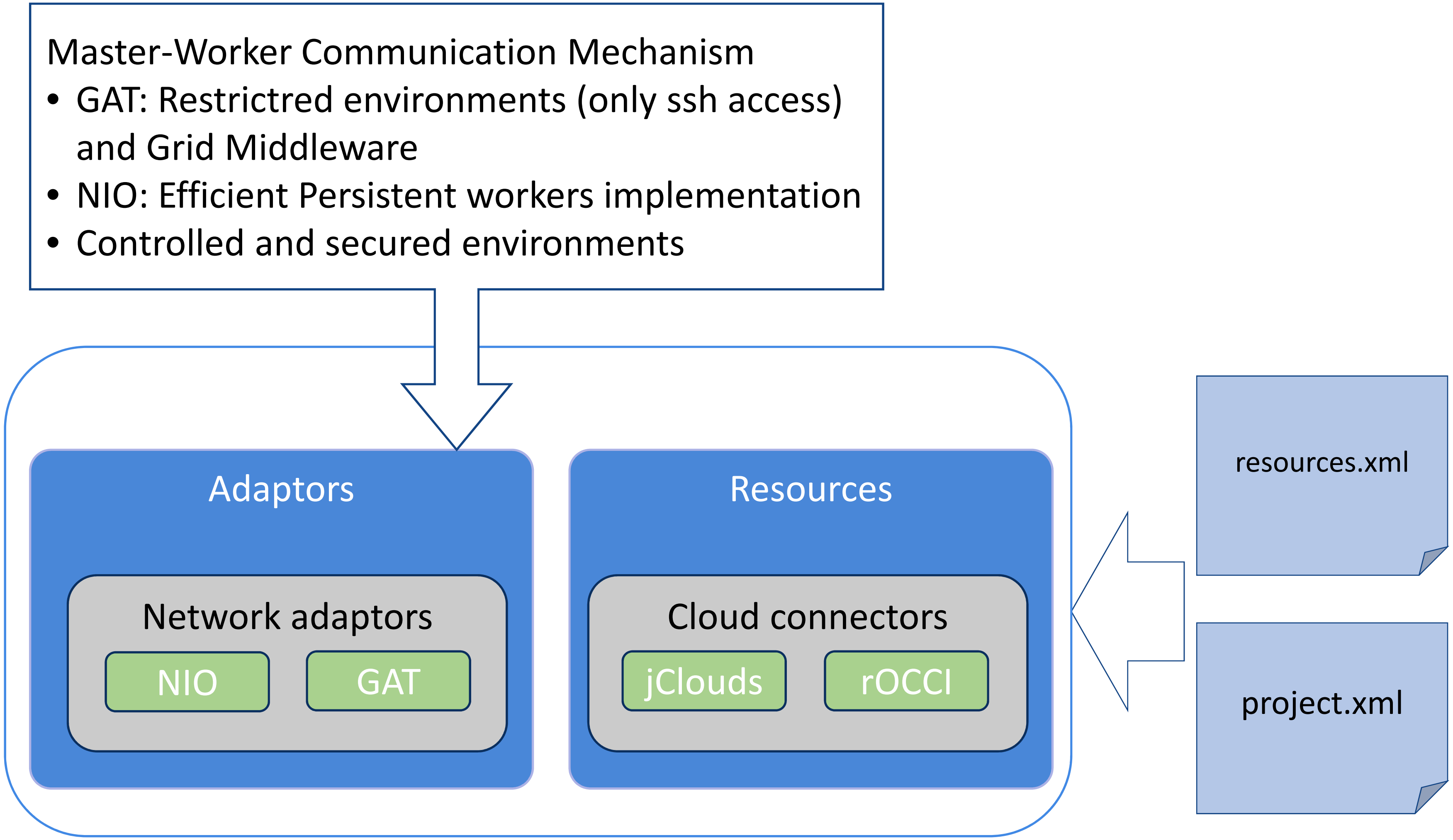


COMPSs Execution Environment

Application Execution Description
- Selection of resources
- Application Code Location
- Working directory



COMPSs Execution Environment



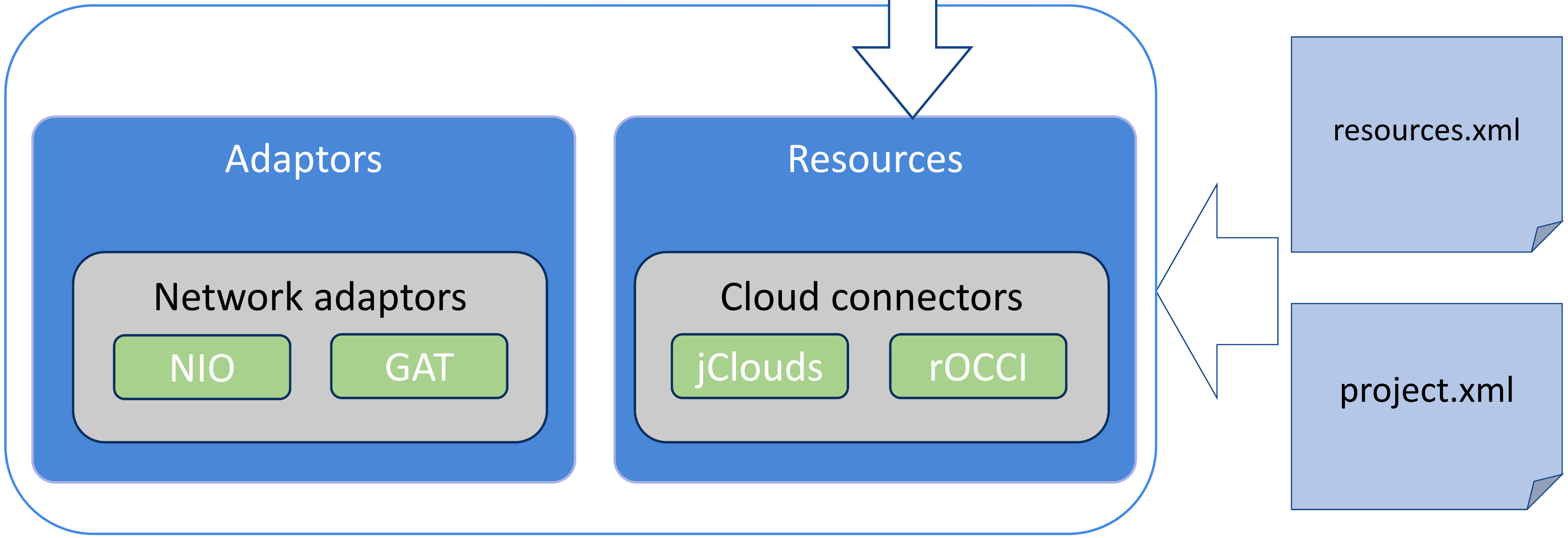
Master-Worker Communication Mechanism

- GAT: Restrictred environments (only ssh access) and Grid Middleware
- NIO: Efficient Persistent workers implementation
- Controlled and secured environments

COMPSs Execution Environment

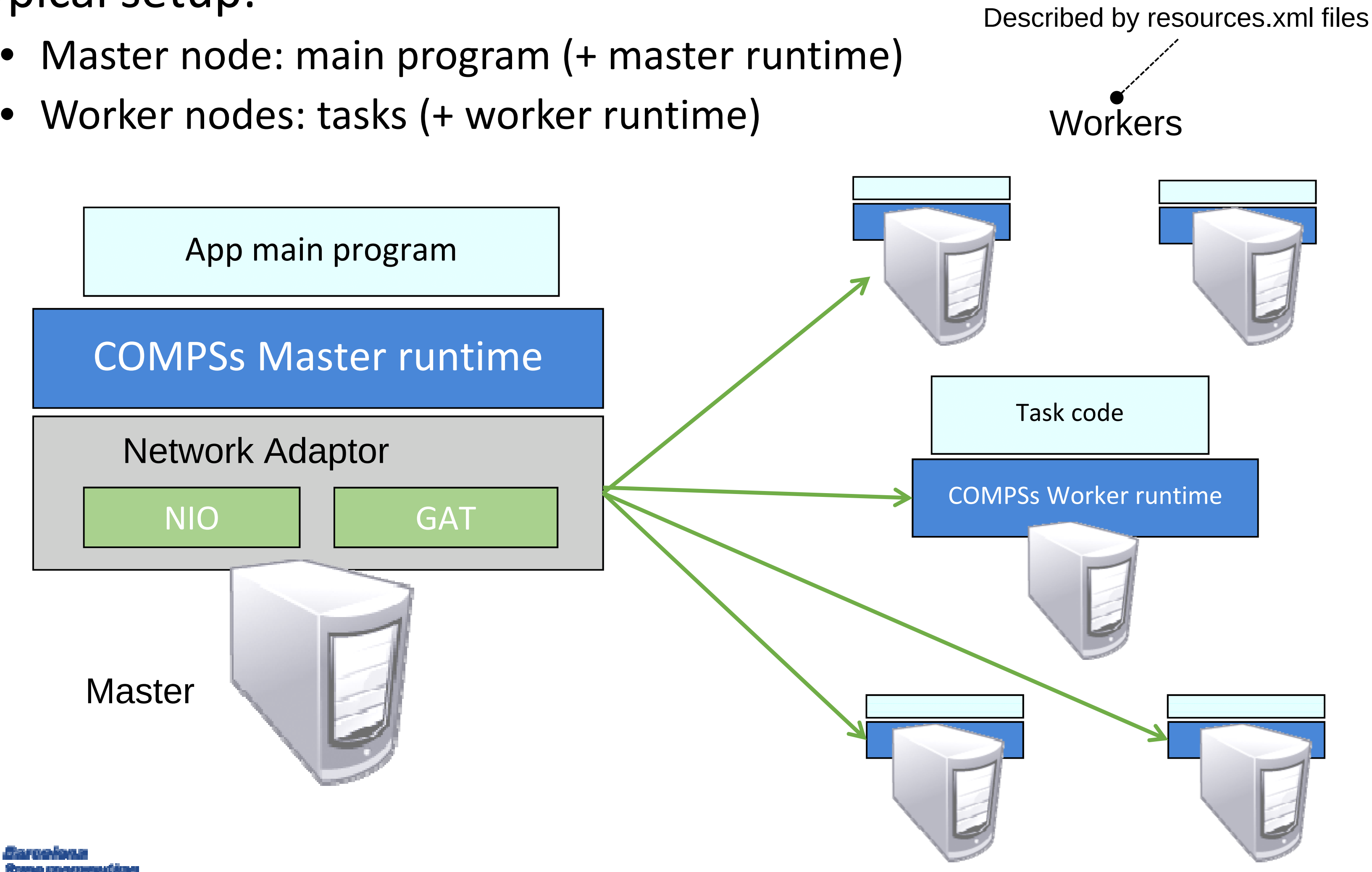
Resource Scalability

- jClouds: access to most of commercial public clouds
- rOCCI: OGF standard
- Extensible (support others..), CIMI in mF2C?



COMPSs in remote hosts (interactive)

- Typical setup:
  - Master node: main program (+ master runtime)
  - Worker nodes: tasks (+ worker runtime)



Constraints matching

- Constraints matching mechanism
  - Enables to choose the optimal resource for each task type
- Applications describe constraints with constraint interface
- The resources description indicates resources available in each host
- Runtime does the matching before doing scheduling

Constraints matching examples

Python decorator

```
@constraint(ComputingUnits=“8”)
@task(A=INOUT, priority=True)
def potrf(A):
    A.dpotrft(lower=True)
```

Java annotations

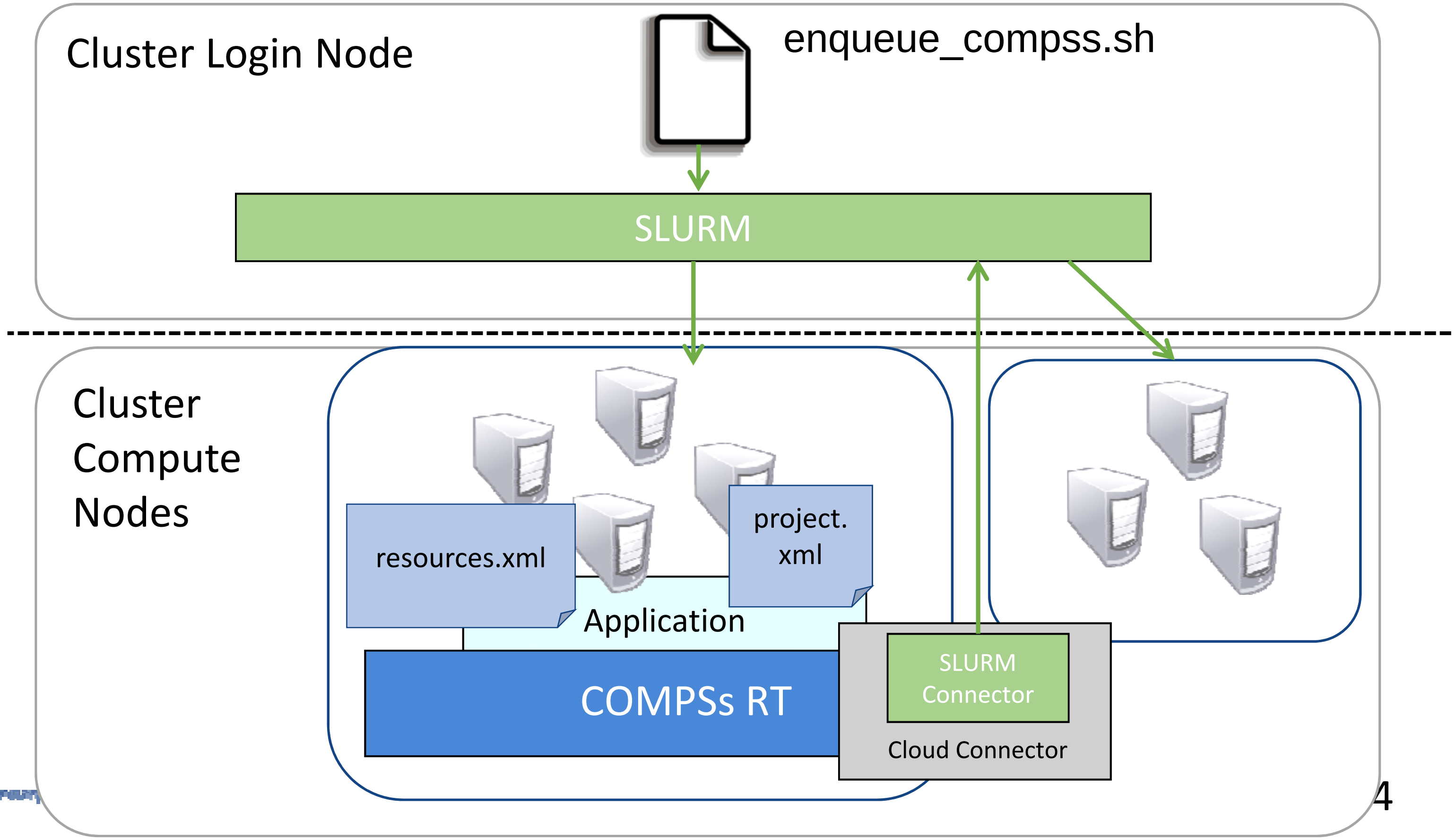
```
@Method(declaringClass = "matmul.files.MatmulImpl")
@Constraints(memorySize="${MIN_MEM_REQ}")
Integer multiplyAccumulativeNative(
    @Parameter() int bsize,
    ...
```

Resource.xml

```
<ComputeNode Name=“172.20.200.18">
  <Processor Name="P1">
    <ComputingUnits>4</ComputingUnits>
    <Architecture>amd64</Architecture>
    <Speed>3.0</Speed>
  </Processor>
 <Memory>
    <Size>256.2</Size>
    <Type>Non-volatile</Type>
  </Memory>
  <Storage>
    <Size>2000.0</Size>
  </Storage>
   <OperatingSystem>
    <Type>Linux</Type>
    <Distribution>OpenSUSE</Distributi
```

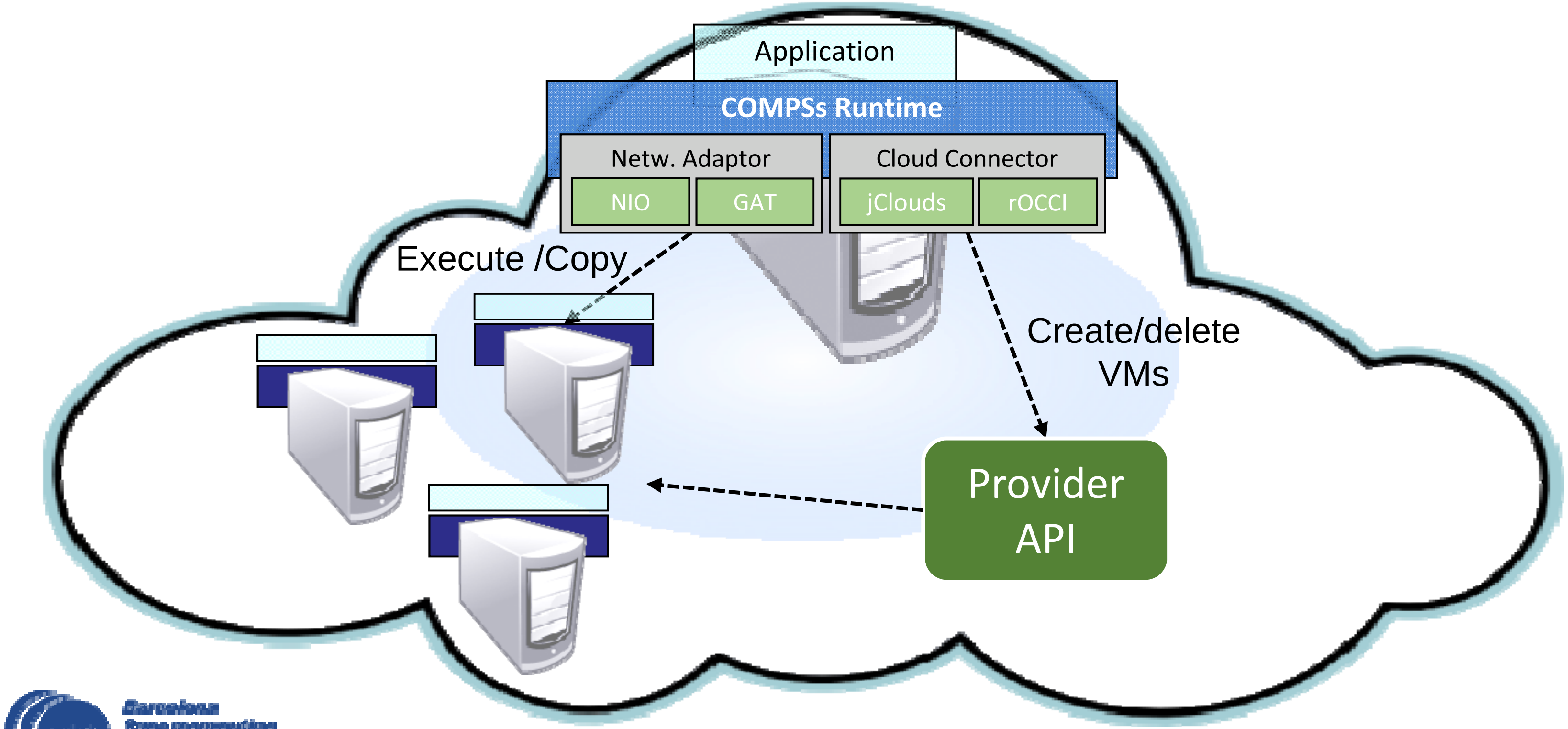
COMPSs in a Cluster (with job scheduler)

- Execution divided in two phases
  - Job-submission of a whole COMPSs app execution – runcompss
    - Project.xml and Resource.xml generated automatically
  - Application execution when allocation is obtained

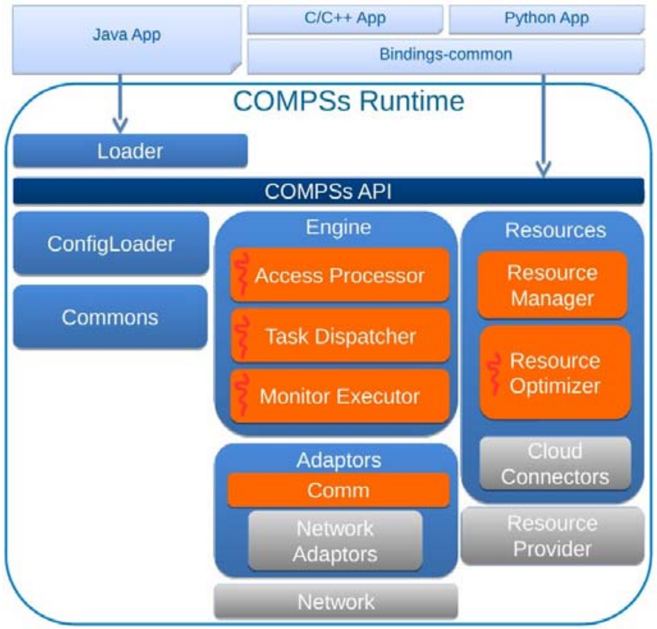


COMPSs in Clouds

- Execution of COMPSs applications in Clouds
  - Select de connector to interact with the Cloud provider
  - Adaptor to communicate VMs (NIO if provider supports firewall management, GAT if only ssh)



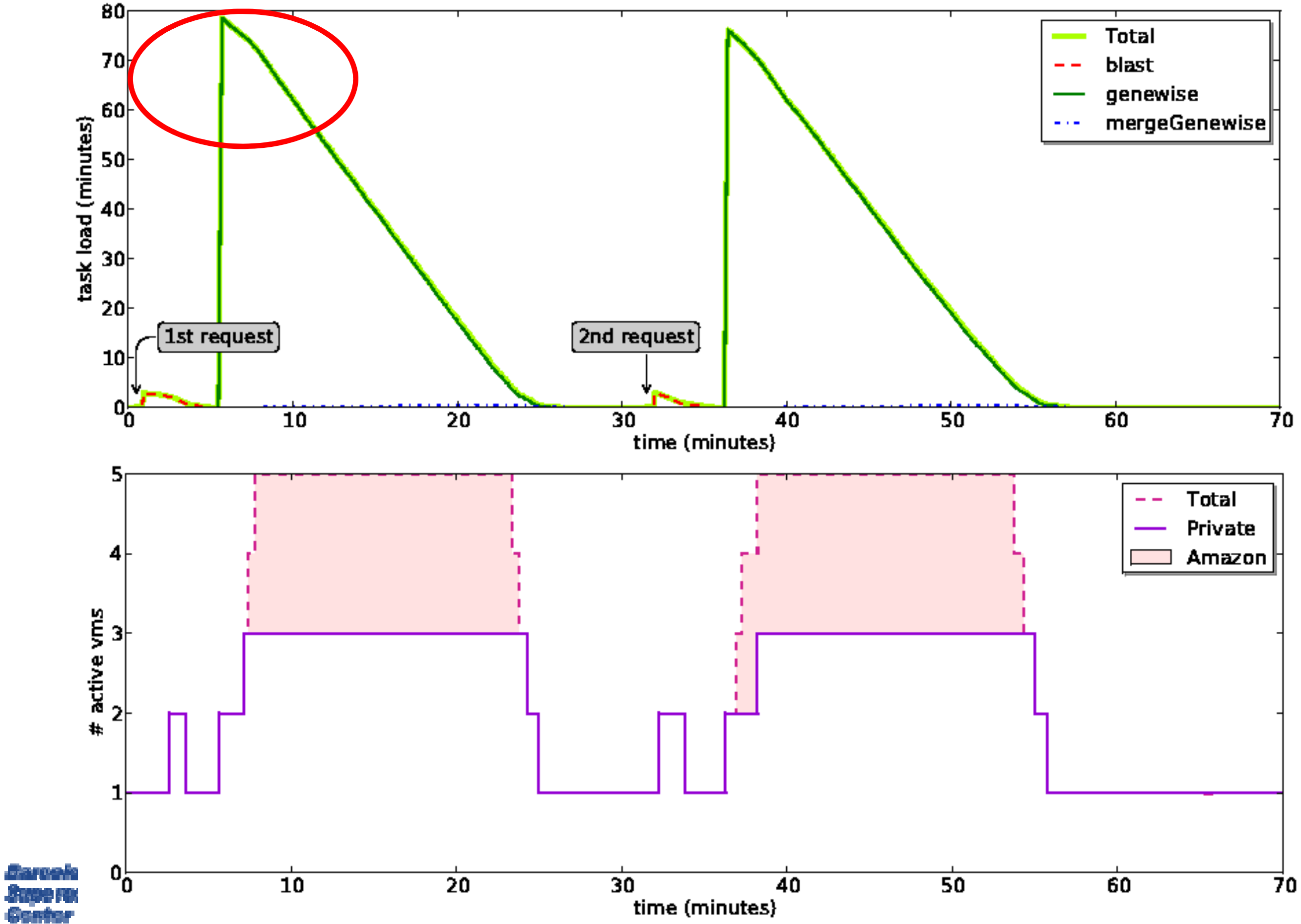
Elasticity in clouds



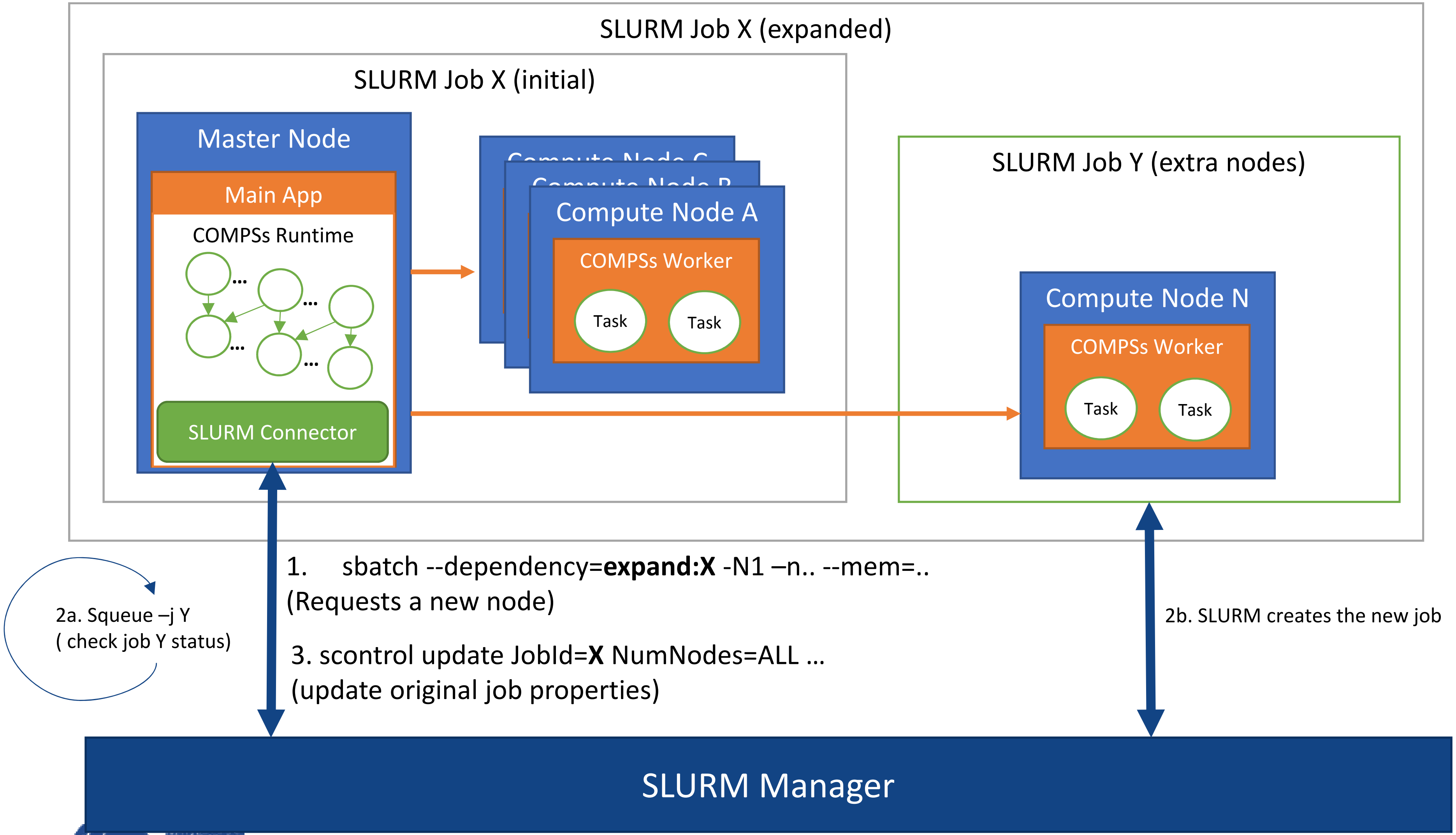
- Acces processor
  - Assigns tasks to VMs or physical resources
- Resource manager
  - Holds resources information (workers and cloud providers)
  - For each Cloud provider, a data structure stores the different available instances (with its features) and the connector that should be used
  - Knows usage of resources
  - Dynamic information
- Resource Optimizer
  - Checks status of workers
  - Can decide
    - To perform load balancing
    - To create/destroy new VMs
  - Sends to Resource Manager requirements about new VM characteristics
  - Evaluates the cloud providers alternatives and chooses the best option
    - More economic
    - The decision can be to open a new private or public VM

Cloud bursting

- Increase/decrease number of VMs depending on task load
- Bursting to Amazon EC2 to face peak load



SLURM Connector (expand)



SLURM Job X (expanded)

SLURM Job X (initial)

Master Node

Main App

COMPSs Runtime

SLURM Connector

Compute Node C

Compute Node B

Compute Node A

COMPSs Worker

Task

Task

SLURM Job Y (extra nodes)

Compute Node N

COMPSs Worker

Task

Task

1. sbatch --dependency=**expand:X** -N1 –n.. --mem=..
(Requests a new node)

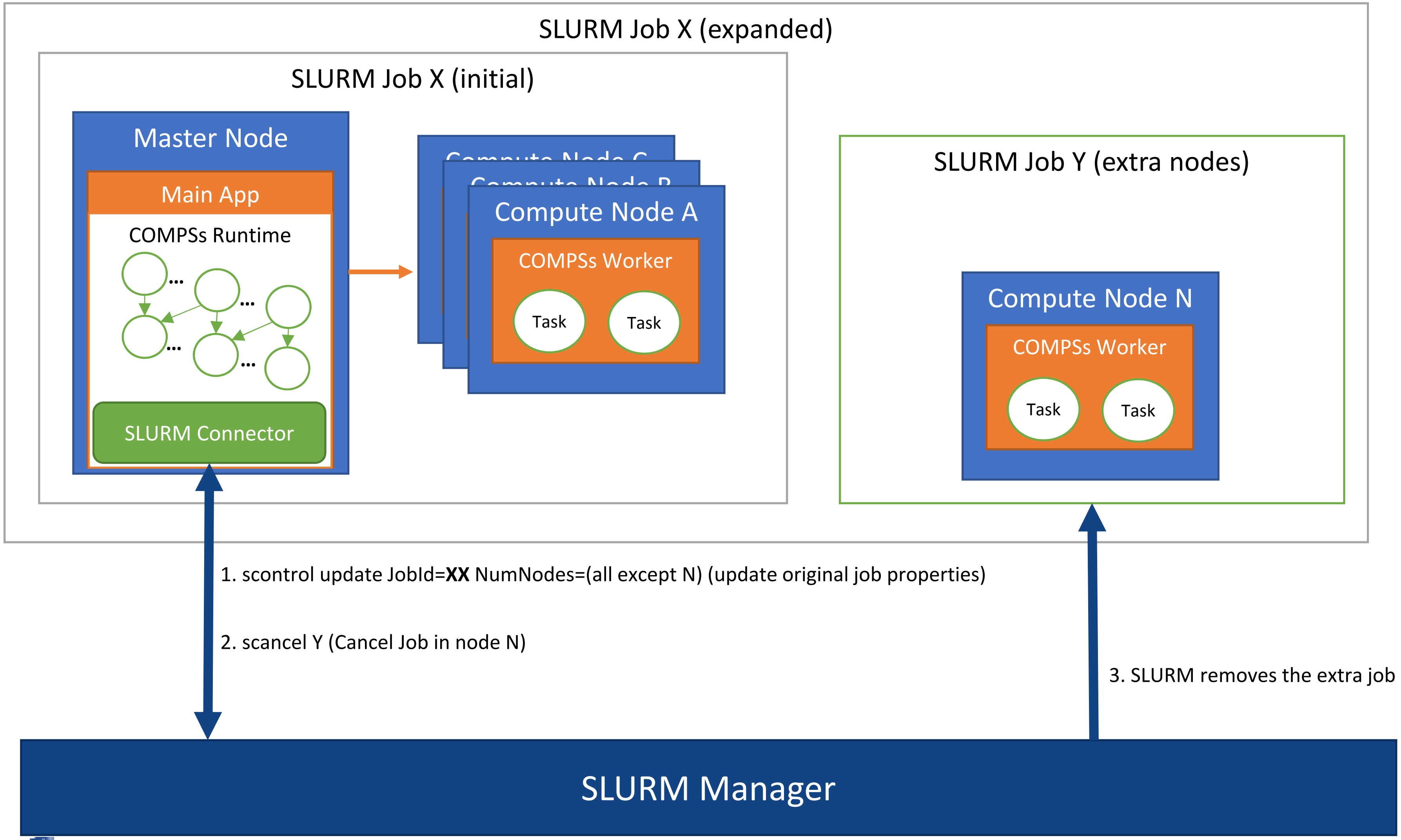
3. scontrol update JobId=**X** NumNodes=ALL …
(update original job properties)

2a. Squeue –j Y
(check job Y status)

2b. SLURM creates the new job

SLURM Manager

SLURM Connector (reduce)



1. scontrol update JobId=**XX** NumNodes=(all except N) (update original job properties)

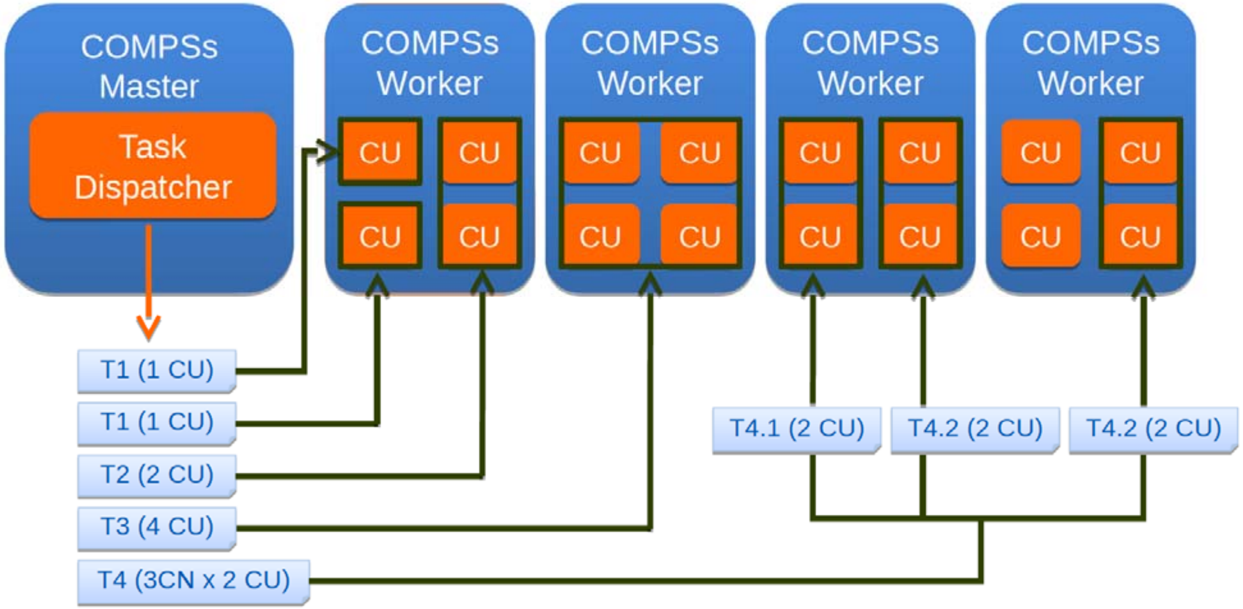
2. scancel Y (Cancel Job in node N)

3. SLURM removes the extra job

Support for MPI tasks

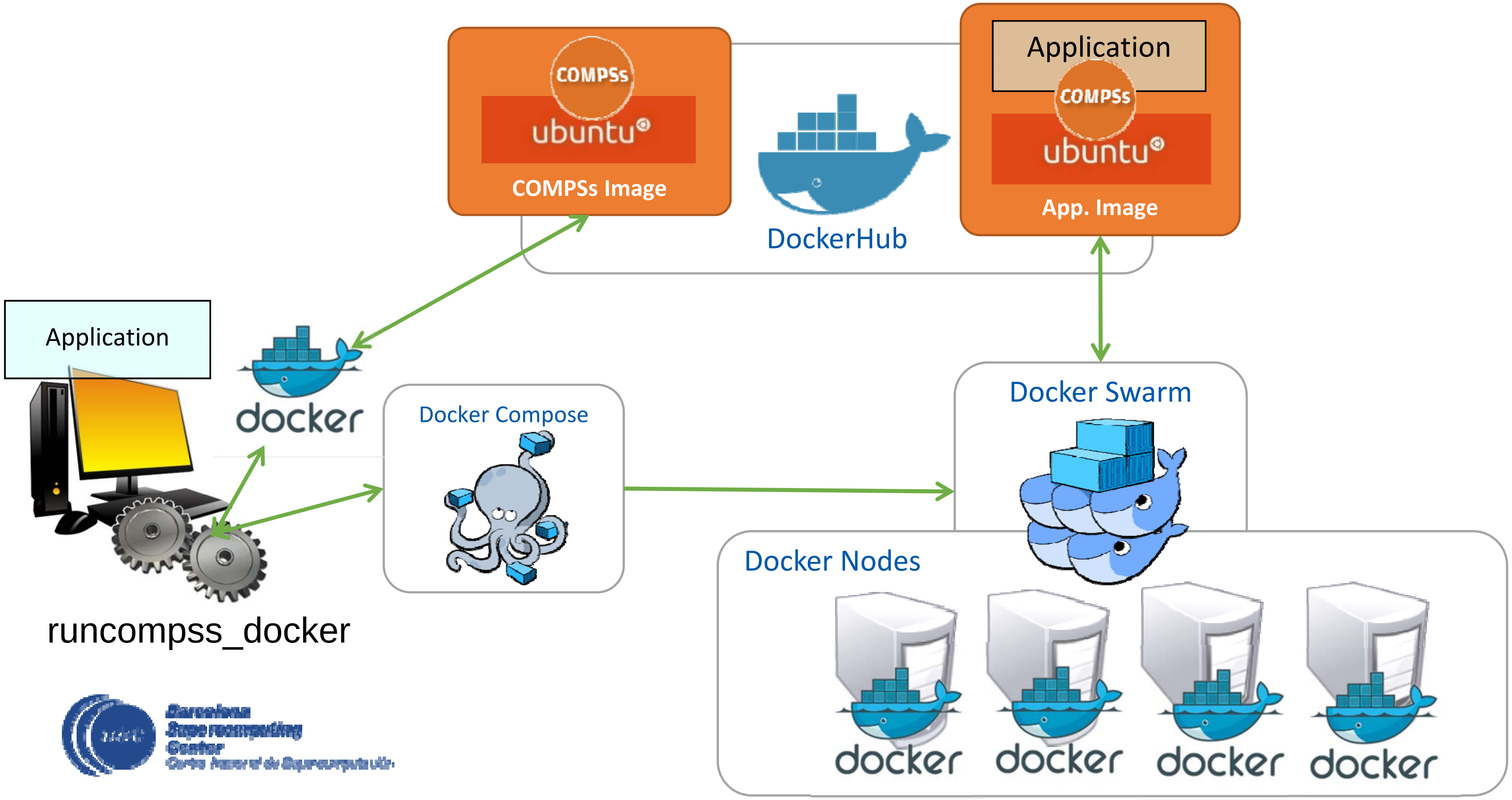
- Extension of interface
- Resource manager aware of multi- node tasks

```
@MPI(mpiRunner = “mpirun”,
           binary = “mpiBinary”,
           computingNodes = “2”,
           workingDir = “/tmp/”,
           priority = “true”,
           constraints = @Constraints(computingUnits = “4”))
void mpiTask();
```



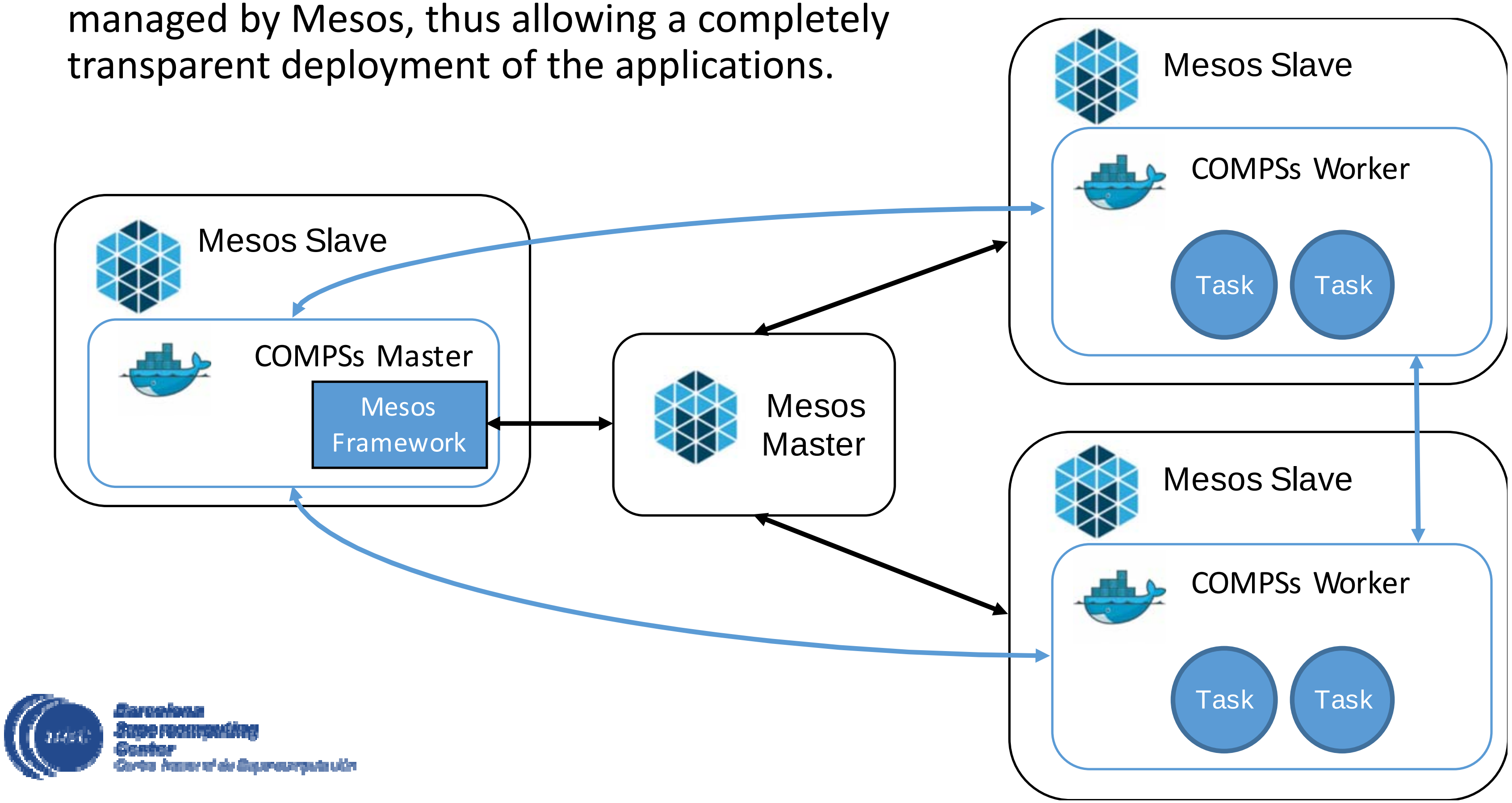
COMPSs with Docker

- Keep as transparent for the user as possible
- Same as running a local COMPSs application (runcompss command)
    - runcompss container
      −−engine=docker −−engine−manager = ’129.114.108.8:4000 ’ −−initial −worker−containers=5
      −−container image=’john123/matmul−example ’ −−classpath=/home/john/matmul/matmul. jar matmul. objects .Matmul 16 4
- Deploy applications as a set of docker container



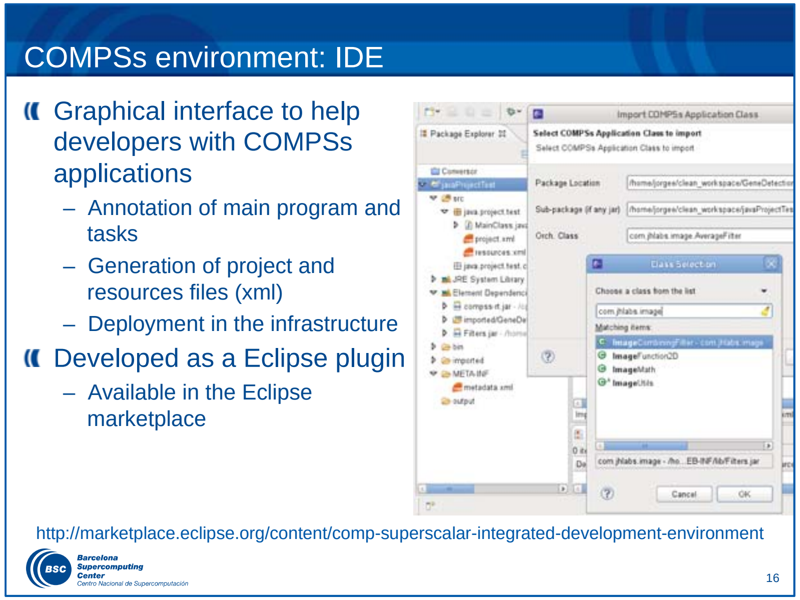
COMPSs with Mesos

- The COMPSs runtime register itself as a Mesos Framework and negotiates the use of resources with the Mesos Master.
- The number and type of nodes requested depends on the actual load.
- Both the COMPSs Master and the workers are executed in Docker containers, managed by Mesos, thus allowing a completely transparent deployment of the applications.



COMPSs development environment

- IDE graphical interface
- Runtime monitor
- Paraver traces

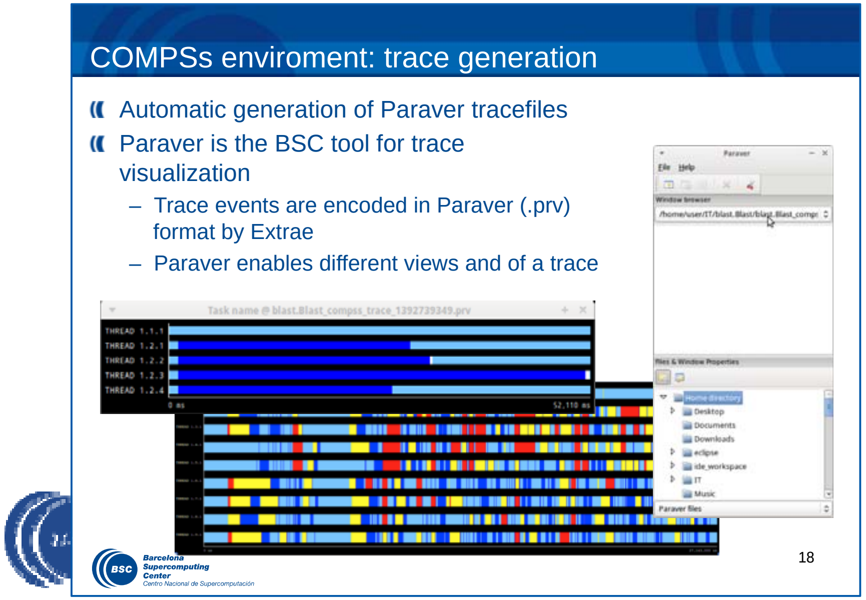


COMPSs environment: IDE

- Graphical interface to help developers with COMPSs applications
  - Annotation of main program and tasks
  - Generation of project and resources files (xml)
  - Deployment in the infrastructure
- Developed as a Eclipse plugin
  - Available in the Eclipse marketplace

http://marketplace.eclipse.org/content/comp-superscalar-integrated-development-environment

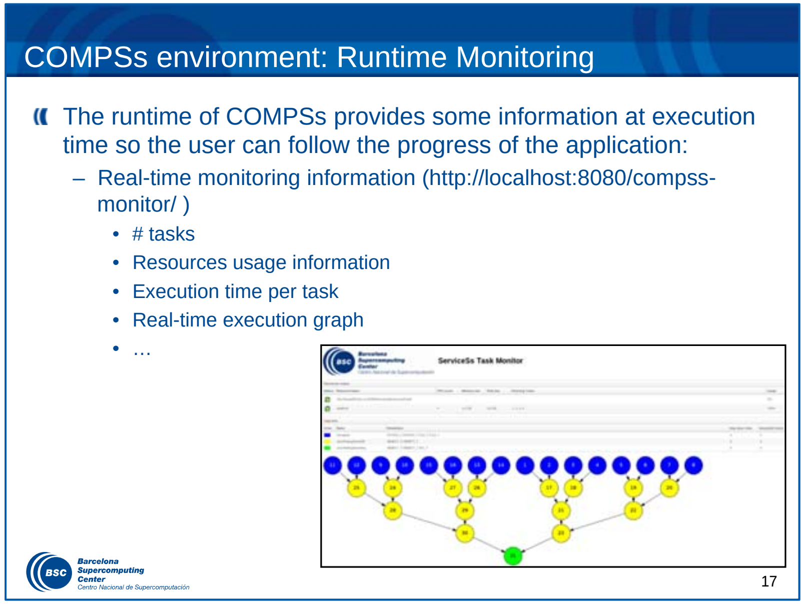
16



COMPSs enviroment: trace generation

- Automatic generation of Paraver tracefiles
- Paraver is the BSC tool for trace visualization
  - Trace events are encoded in Paraver (.prv) format by Extrae
  - Paraver enables different views and of a trace

18



COMPSs environment: Runtime Monitoring

- The runtime of COMPSs provides some information at execution time so the user can follow the progress of the application:
  - Real-time monitoring information (http://localhost:8080/compss-monitor/)
    - # tasks
    - Resources usage information
    - Execution time per task
    - Real-time execution graph
    - …

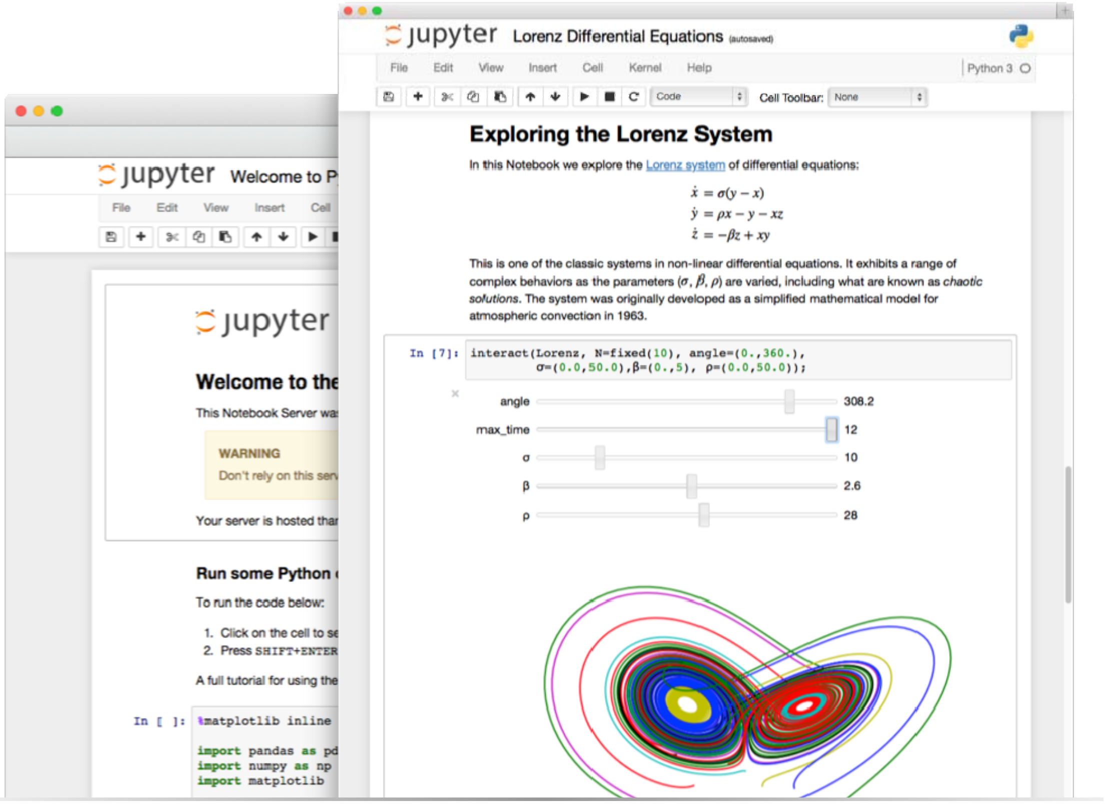
17

New challenges

- New challenges in distributed computing
  - Dynamic workflows
  - Integration with novel storage technologies
    - Hecuba/dataClay
  - Integration of task-based with traditional HPC programming models
    - MPI
    - OmpSs
    - GPU and FPGAs
  - Alternative computing platforms
    - Fog to cloud architectures
    - Mobile computing

Integration with Jupyter notebook

- The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.
- Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.
- Runs Python – sequential
- PyCOMPSs integrated with Jupyter notebook
  - Runs in parallel in local node and can offload tasks to external nodes



jupyter

Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0));
```

angle 308.2
max_time 12
σ 10
β 2.6
ρ 28

Installation

- Release 2.2 – December 2017
- OVA available in downloads with all software installed and examples
- Installation manual:
  - http://compss.bsc.es/releases/compss/latest/docs/COMPSs_Installation_Manual.pdf
- Source code:
  - http://compss.bsc.es/ (Downloads Section – Source)
- Packages and repositories:
  - http://compss.bsc.es/ (Downloads Section – Repository references)
    - Debian based: apt-get install compss-framework
    - Zypper based: zypper install compss-framework
    - Yum based: yum install compss-framework
- Supercomputers:
    - $ wget http://compss.bsc.es/repo/sc/stable/COMPSs_2.0.tar.gz
    - $ tar -xvzf COMPSs_2.0.tar.gz
    - $ cd COMPSs
    - $ ./install <targetDir>
- Pip:
    - sudo -E pip install compss –v
    - source /etc/profile.d/compss.sh

Additional Notes

- Project page:
  - http://www.bsc.es/compss
- Direct downloads page:
  - http://www.bsc.es/computer-sciences/grid-computing/comp-superscalar/download
    - Virtual Appliance for testing & sample applications
    - Tutorials
    - Red-Hat & Debian based installation packages
    - Source Code
- Application Repository
  - http://compss.bsc.es/projects/bar/wiki/Applications
    - Several examples of applications developed with COMPSs

www.bsc.es

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

Thank you

For further information please contact
daniele.lezzi@bsc.es